УДК 681.3

М.К. Буза
Белорусский государственный университет, г. Минск
bouza@bsu.by

Инструментальные средства проектирования параллельных программ

Проанализированы современные средства проектирования параллельных программ. Выявлены свойства циклов в программах и создана их классификация. Предложена функциональная библиотека распараллеливания циклов. Разработана общая схема преобразования последовательных программ в последовательно-параллельные.

Введение

Пиковая производительность многопроцессорных вычислительных систем (МВС) на реальных задачах существенно ниже. Наряду с техническими проблемами снижения производительности существенное влияние на нее оказывает несоответствие архитектурных решений компьютеров алгоритмам решения задач и средствам описания параллелизма, так называемая проблема семантического разрыва.

Для частичного преодоления такого разрыва необходимо разрабатывать автоматизированные (а лучше автоматические) системы проектирования параллельной обработки. Это позволит приблизить структуру решения задачи соответствующим архитектурным решениям компьютеров. Тогда многие проблемы по повышению технических характеристик МВС не станут так актуальны.

Параллельный подход к проектированию программ требует решения ряда дополнительных проблем, среди которых: порождение и обработка синхронных событий, параллельных потоков, организация эффективного планирования и т.д.

На сегодняшний день созданы различные средства для проектирования и описания параллельных процессов [1]. Разработаны специальные примитивы коммуникации и управления, позволяющие программисту создавать сценарии параллельного исполнения программ; имеются расширения последовательных языков специальными конструкциями, порождающими параллельные процессы (например, mpC, HPF, HPC++, Cilk, Maisie, MPL и др.); созданы языки параллельного программирования (Java, NESL, Orca, Sisal, ZPL и др.).

С другой стороны, можно использовать специальные библиотеки (например, PVM, MPI и т.д.), позволяющие организовывать асинхронный обмен сообщениями между потоками и процессами [2].

Организовать параллельную обработку можно, используя системные функции операционной системы, такие, например, как Portable Operating System Interface for UNIX (POSIX). Это стандарты, описывающие интерфейсы между ОС и прикладной программой. В пределах POSIX параллельные вычисления реализуются на основе обмена сообщениями (как в MPI) или разделяемой памяти (как в OpenMP).

Получила поддержку идея «инкрементального распараллеливания», где принятие решения о последовательном или параллельном исполнении программы возлага-

ется на компилятор, а вместо создания параллельной программы пользователь добавляет в исходный код специальные директивы, облегчающие работу компилятора.

Однако ограничения на скорость передачи информации (распространение сигнала на кристалле), проблемы охлаждения и потребления энергии привели к идее многоядерных процессоров, основанных на принципе близкодействия. Это существенно упростило объем управляющей логики. Сегодня уже реализован 48-ядерный кристалл Single-Chip Cloud Computer, кристалл Tile [3], содержащий сотню процессорных ядер. Таким образом, проблема тактовой частоты не стоит так остро, как раньше.

Создание многоблочной памяти с расслоением и организация потока запросов к ней при наличии механизма отложенных обращений к памяти в разумных пределах не станет заметным ограничением на пути повышения производительности вычислений. Безусловно, программы для обработки должны быть подготовлены в виде совокупности синхронных и асинхронных тредов.

Общий недостаток использования указанных средств состоит в том, что большой объем работы по организации параллельной обработки возлагается на программиста. В связи с этим возникает задача минимизации ручной работы при проектировании параллельных программ и организации параллельной обработки данных.

Разрабатывать параллельные программы вручную, начиная с формализации постановки задачи, труд необычайно сложный. Этот процесс следует автоматизировать.

Целью статьи является разработка формальных свойств циклов в программах, на основе которых осуществляется их распознавание и кластеризация, создание алгоритмов их распараллеливания и библиотеки программ для их реализации. Это позволяет разработать общую схему трансфера последовательных программ в последовательно-параллельные.

Постановка задачи

На сегодняшний день накоплено огромное количество компьютерных программ для решения разнообразных задач в последовательном режиме. Это отлаженные и неоднократно апробированные программы. Среди них достаточное множество задач, требующих большого объема вычислений. К ним можно отнести задачи моделирования результатов ядерных испытаний, обработки аэрокосмической информации, сейсморазведки и многие другие.

Многие задачи, которые позиционируются, как сложные, требующие суперкомпьютеров для их обработки, не всегда таковыми являются. Надо тщательно разрабатывать вычислительные модели, понижая порядок, стремясь сделать их одноранговыми.

В связи с этим актуальной становится задача трансфера программ для их решения из последовательной формы в последовательно-параллельную.

Ниже предлагается алгоритм исследования исходных параллельных программ, выделение в них наиболее трудоемких для вычисления циклов и распараллеливания их с последующим планированием и оценкой сложности их обработки в параллельном режиме. Предложена также общая схема автоматизированной системы распараллеливания программ.

Автоматизированная система проектирования параллельных программ должна включать: анализ исходной программы, определение линейных участков, фрагментов с ветвлением и циклические участки. Затем для каждого из фрагментов определяются различные типы зависимости, после чего идет распараллеливание исходной программы.

Схема распараллеливания программ

Общая схема автоматизированной системы проектирования параллельных программ может быть представлена в следующем виде (рис. 1).

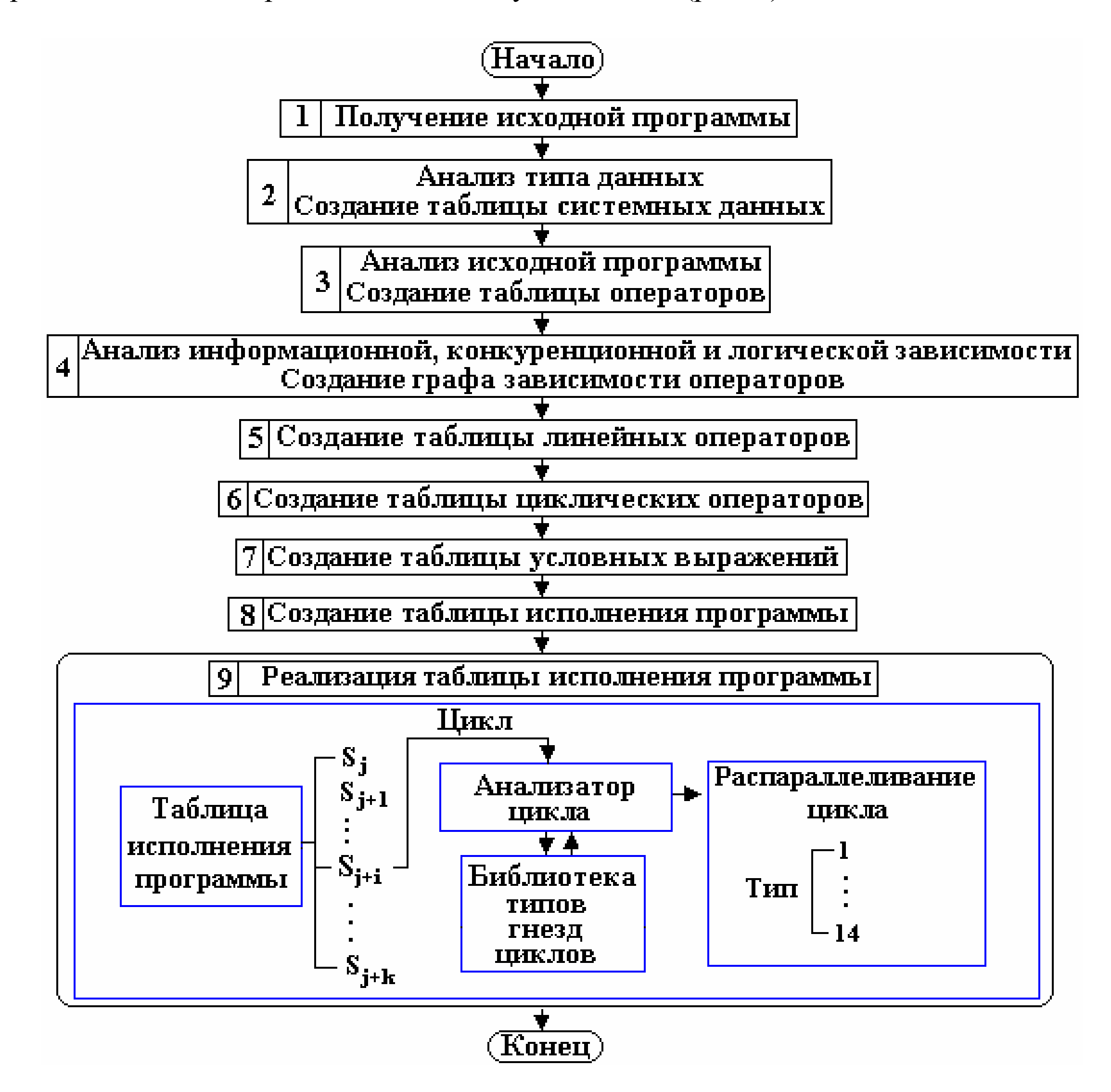


Рисунок 1 – Общая схема распараллеливания программ

Реализация представленного на рис. 1 алгоритма осуществляется следующим образом.

Шаг 1. Открываем текстовый файл и записываем исходную программу в одномерный массив.

Шаг 2. Анализируем тип используемых данных. Затем их начальные значения, имена и типы данных записываем в таблицу системных данных.

Рассмотрим реализацию шага 2 на примере анализа констант (рис. 2).

Пусть задана константа типа «#define N 100». Если ключевым словом первой строки является «#define», оно определяет константу. Выделяется ключевое слово первой строки – «#define». За ключевым словом следует имя данных – N. За именем данных – его значение: 100.

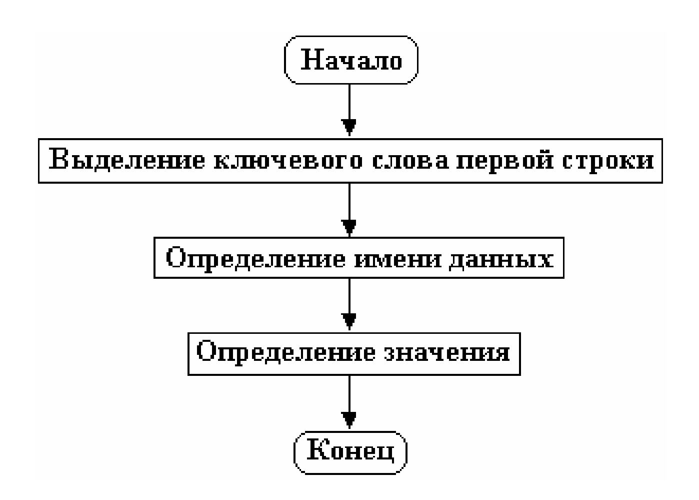


Рисунок 2 – Алгоритм анализа констант

Шаг 3. Анализируем исходную программу и создаем таблицу операторов. Она содержит поля: номер оператора, оператор, тип оператора, принадлежность оператора к телу цикла и глубина вложения.

Шаг 4. Определяем информационную логическую и конкуренционную зависимости операторов.

Шаг 5. Формируем таблицу линейных операторов. Она включает следующие поля: номер оператора в программе, оператор, левая часть до знака «=» и правая часть после данного знака.

Шаг 6. Формируем таблицу циклических операторов. Она имеет следующую структуру: номер оператора в таблице, номер оператора в программе, индекс цикла, начальное и конечное значение переменной цикла, шаг изменения цикла и тело цикла.

Шаг 7. Создаем таблицу условных операторов. Ее структура аналогична структуре таблицы циклических операторов.

Шаг 8. Формируем таблицу исполнения программы на основе подготовленных выше таблиц.

Шаг 9. Формируем параллельную программу.

Шаг 10. Распределяем работу по процессорам в соответствии с принятой стратегией.

Алгоритм распараллеливания циклов

Так как основное вычислительное время занимает обработка циклов, то естественно в первую очередь следует разработать и реализовать алгоритм их распараллеливания.

Циклы в программах различаются по способам их организации, по структуре тела цикла, по глубине вложенности, по механизмам использования переменных и констант и многим другим характеристикам. Ясно, что указанные особенности циклов могут оказать существенное влияние на процесс проецирования общего алгоритма на конкретный тип цикла в программе и последующий алгоритм распараллеливания.

В соответствии с указанными характеристиками выделены четырнадцать типов циклов, описаны их свойства, создана система распознавания типа цикла, разработана библиотека, позволяющая проецировать общий алгоритм распараллеливания на

конкретный тип цикла с последующим планированием и обработкой его в параллельном режиме на мультипроцессорной системе.

Общий алгоритм распараллеливания циклов показан на рис. 3.

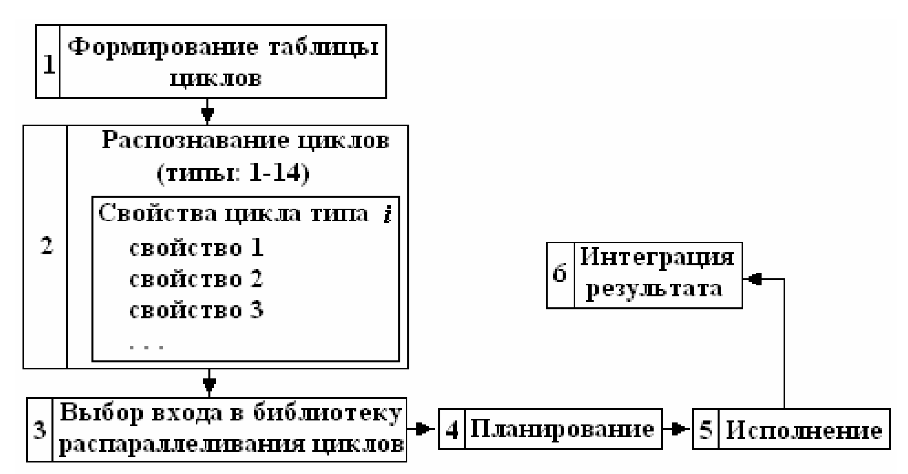


Рисунок 3 – Алгоритм обработки циклов

Приведем примеры некоторых типов циклов.

Тип 1. Он характеризуется следующим:

1. В каждый цикл не входит переменная кроме индекса цикла.

Например, “for(i=0;i<(цифра);i++){” ← цикл без переменных.

2. В теле самого внутреннего цикла находится линейный оператор, который имеет вид: sum = sum ⊕ (цифра) ← константа; где sum – переменные с типом данных “double”, “float”, “long” или “int”; ⊕ – арифметическая операция.

Тип 2. Цикл типа 2 характеризуется следующим:

1. В каждый цикл не входит переменная кроме индекса цикла.

2. В теле самого внутреннего цикла находится линейный оператор, который имеет вид:

$$\text{sum} = \text{sum} \oplus (\text{переменная}); \leftarrow \text{переменная} \quad (1)$$

Тип 3. Цикл типа 3 характеризуется следующим:

1. В каждый цикл входят переменные.

Например,“for(i=(переменная);i<(переменная);i++){” ← цикл с переменными.

2. В теле самого внутреннего цикла находится линейный оператор вида (1).

Тип 4. Особенность цикла в том, что в теле цикла находится линейный оператор следующего вида: “sum = *f*(sum, i);”← линейная функция.

Функция *f*(sum, i) имеет вид: *f*(sum, i) = sum ⊕ (a × i), где $a \neq 0$; i – индекс цикла.

Дальнейшие особенности касаются структур тела циклов. Рассмотрим особенности реализации тела цикла на реальных примерах.

Тип 5. Этот тип цикла можно встретить в программе для нахождения максимального и минимального значения в массиве.

Последовательная программа нахождения максимального значения в массиве может быть записана следующим образом:

```
for(i=(переменная);i<(переменная);i++){ ← цикл с переменными.
    if (s1<arr[i]){ (или if (arr[i]> s1) ) ←  условный оператор находится в теле цикла.
        s1 = arr[i]; ← линейный оператор находится в теле условного оператора.
}  }
```

где s1 – максимальное значение в одномерном массиве.

Цикл типа 5 характеризуется следующим:

1. Цикл с переменными.

2. В теле цикла находится условный оператор, который включает в себя переменную и массив, где переменная служит для хранения максимального или минимального значения в массиве.

3. В теле условного оператора находится линейный оператор. Переменная находится в его левой части, массив находится в правой части, то есть входное множество – массив.

Оценка времени вычисления параллельных программ

Время выполнения программы на однопроцессорной системе ($T_{S_ExecuteProgram}$) имеет вид:

$$T_{S_ExecuteProgram}=T_{S_Initial\text{-}program}+T_{S_ExePro_Part\text{-}of\text{-}no\text{-}loop}+T_{S_ExePro_Part\text{-}of\text{-}loop},$$

$$T_{S_ExePro_Part\text{-}of\text{-}no\text{-}loop}=N_{Expression}\times T_{Computing_expression}+N_{if_else}\times T_{Computing_if_else},$$

$$T_{S_ExePro_Part\text{-}of\text{-}loop}=N_{S\text{-}Loop}\times T_{S_Computing_for},$$

где $T_{S_Initial\text{-}program}$ – время инициализации программы для вычислений на однопроцессорной системе; $T_{S_ExePro_Part\text{-}of\text{-}no\text{-}loop}$ – время выполнения автономных линейных и условных операторов, которые не вложены в гнездо циклов; $T_{S_ExePro_Part\text{-}of\text{-}loop}$ – время выполнения циклических операторов; $N_{Expression}$ – количество автономных линейных операторов в программе; N_{if_else} – количество автономных условных операторов в программе; $N_{S\text{-}Loop}$ – количество автономных циклических операторов в программе.

Время выполнения программы на параллельной системе ($T_{M_ExecuteProgram}$) имеет вид:

$$T_{M_ExecuteProgram}=T_{M_Initial\text{-}program}+T_{M_ExePro_Part\text{-}of\text{-}no\text{-}parallel}+T_{M_ExePro_Part\text{-}of\text{-}parallel},$$

$$T_{M_ExePro_Part\text{-}of\text{-}no\text{-}parallel}=N_{Expression}\times T_{Computing_expression}+N_{if_else}\times T_{Computing_if_else}+$$
$$+N_{M\text{-}Loop\text{-}of\text{-}noParallel}\times T_{S_Computing_for},$$

$$T_{M_ExePro_Part\text{-}of\text{-}parallel}=N_{M\text{-}Loop\text{-}of\text{-}parallel}\times T_{M_Computing_for},$$

где $T_{M_Initial\text{-}program}$ – время инициализации программы для вычислений на параллельной системе; $T_{M_ExePro_Part\text{-}of\text{-}no\text{-}parallel}$ – время выполнения части последовательных исполнений в программе; $T_{M_ExePro_Part\text{-}of\text{-}parallel}$ – время выполнения части параллельных исполнений в программе; $N_{M\text{-}Loop\text{-}of\text{-}noParallel}$ – количество автономных циклических операторов, которые не могут быть выполнены параллельно; $N_{M\text{-}Loop\text{-}of\text{-}parallel}$ – количество автономных циклических операторов, которые могут быть выполнены параллельно.

Ускорение вычисляется по формуле:

$$R = T_{S_ExecuteProgram} / T_{M_ExecuteProgram}.$$

Рассмотрим ускорение вычислений при различном количестве процессоров на конкретной задаче.

Графики, соответствующие выполненному эксперименту для численного решения уравнения $f(x)=M$ при условии: $M > 0$, $M > x_1 \geq 1$, $M > x_2 \geq 1$, ..., $M > x_n \geq 1$ для конкретных значений параметров, представлены на рис. 4 и 5.

Проведенные исследования показывают, что, во-первых, при одинаковом количестве обрабатывающих процессоров с увеличением количества итераций среднее

время вычисления растет. При возрастании количества обрабатывающих процессоров среднее время вычисления уменьшается.

Во-вторых, если количество итераций существенно увеличивается, каждый обрабатывающий процессор приближается к среднему времени вычисления, поэтому достигается лучшая производительность системы.

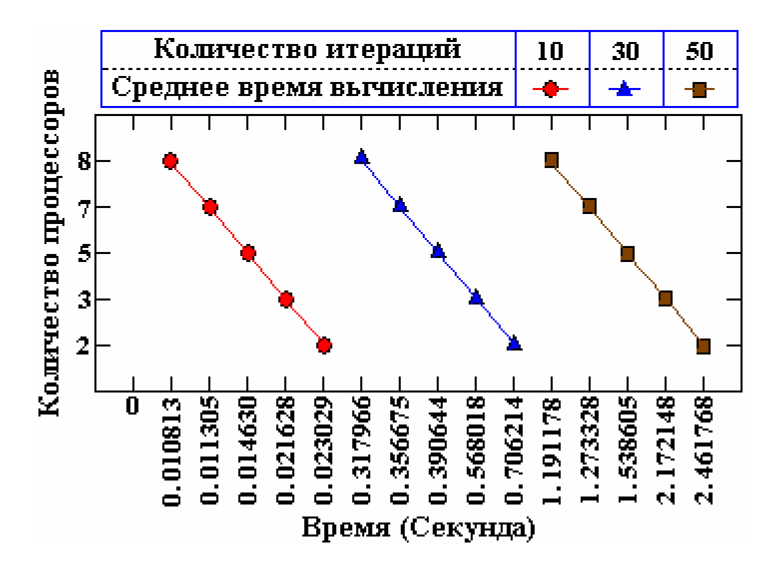


Рисунок 4 – Среднее время вычисления неопределенного уравнения на кластере типа Beowulf с сетью Fast Ethernet

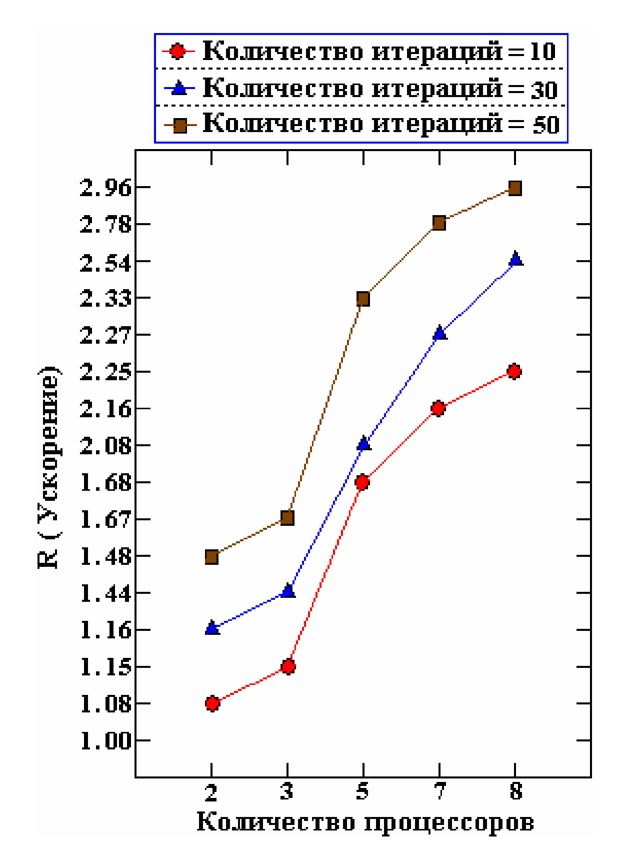


Рисунок 5 – Зависимость ускорения вычислений неопределенного уравнения от количества процессоров

Предложенный алгоритм и поддерживающая его автоматизированная система были апробированы на различных задачах, в частности, для распараллеливания задачи логического отображения (logistic map) [4], где подтвердила свою работоспособность и эффективность.

Выводы

Предложенный метод распараллеливания позволяет преобразовывать исходные последовательные программы, содержащие циклы рассматриваемого типа, для параллельных вычислений, что дает возможность использовать накопленные для последовательного исполнения информационно-программные ресурсы на МВС без ручного перепрограммирования.

Из приведенных графиков видно, что увеличение количества итераций увеличивает общее время решения задач при одинаковом количестве обрабатывающих процессоров.

Созданный метод формирования свойств циклов на основе способов их организации, структуры тела цикла, глубины вложенности, механизма использования переменных и констант позволил осуществлять их распознавание в программах и кластеризацию.

Применение предложенного алгоритма распараллеливания гнезд циклов вместе с разработанной библиотекой циклов позволяет создавать параллельную структуру циклов, даже если они включают вызовы других подпрограмм. Эффективная реализация схемы автоматизации распараллеливания программ и успешный опыт анализа реальных программ доказывают высокую продуктивность предложенного подхода.

Литература

1. Буза М.К. Системы параллельного действия / Буза М.К. – Минск : Издательский центр БГУ, 2009. – 415 с.
2. Буза М.К. Архитектура компьютеров / М.К. Буза // Учебник для университетов. – Изд-во «Новое знание», 2007. – 559 с.
3. Wentzlaff D. On-Chip interconnection Architecture of the Tile Processor / D. Wentzlaff // IEEE Micro. – September-October, 2007.
4. Буза М.К. Реализация логического отображения на системе параллельного действия / Буза М.К., Цзяхуэй Лю // Информатика. – 2009. – № 2. – С. 131-140.

М.К. Буза
Інструментальні засоби проектування паралельних програм
Проаналізовані сучасні засоби проектування паралельних програм. Виявлені властивості циклів в програмах і створена їх класифікація. Запропонована функціональна бібліотека розпаралелювання циклів. Розроблена загальна схема перетворення послідовних програм в послідовно-паралельні.

M.K. Buza
Software Tools for Design Parallel Program
Properties cycles in programs are formalized and classification cycles are given. Function program library parallel cycles design is done. Common schema transformation program from seguental form to seguental-parallel is suggested and described.

Статья поступила в редакцию 10.06.2010.